Business Case: LoanTap Logistic Regression

Problem statement

LoanTap, an online lending platform catering to millennials, seeks to optimize its underwriting process for Personal Loans. The data science team's objective is to assess the creditworthiness of individuals and decide on extending credit lines. The focus is on delivering instant, flexible loans with consumer-friendly terms to salaried professionals and businessmen. The challenge lies in analyzing various attributes to make informed decisions on eligibility and recommend personalized repayment terms. This initiative aligns with LoanTap's commitment to innovation in the loan sector, ensuring efficient and tailored financial solutions for both MSMEs and individuals in the form of Personal Loans, EMI Free Loans, Personal Overdrafts, and Advance Salary Loans.

In [1]:
```
#import all libraries
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/00
```

In [4]:
```
df

# Dataset has 396030 rows and 27 columns
```

Out[4]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	h
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	
...	...	...	...	...	...	...	...	...	
396025	10000.0	60 months	10.99	217.38	B	B4	licensed bankere	2 years	
396026	21000.0	36 months	12.29	700.42	C	C1	Agent	5 years	
396027	5000.0	36 months	9.99	161.32	B	B1	City Carrier	10+ years	
396028	21000.0	60 months	15.31	503.02	C	C2	Gracon Services, Inc	10+ years	
396029	2000.0	36 months	13.61	67.98	C	C2	Internal Revenue Service	10+ years	

396030 rows × 27 columns

In [5]:

```
df.info()

# It appears to be missing values in some of the features
# Datatype of all features are either float64 or object
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

1. **loan_amnt** : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
2. **term** : The number of payments on the loan. Values are in months and can be either 36 or 60.
3. **int_rate** : Interest Rate on the loan
4. **installment** : The monthly payment owed by the borrower if the loan originates.
5. **grade** : LoanTap assigned loan grade
6. **sub_grade** : LoanTap assigned loan subgrade
7. **emp_title** :The job title supplied by the Borrower when applying for the loan.*
8. **emp_length** : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
9. **home_ownership** : The home ownership status provided by the borrower during registration or obtained from the credit report.
10. **annual_inc** : The self-reported annual income provided by the borrower during registration.
11. **verification_status** : Indicates if income was verified by LoanTap, not verified, or if the income source was verified
12. **issue_d** : The month which the loan was funded
13. **loan_status** : Current status of the loan - Target Variable
14. **purpose** : A category provided by the borrower for the loan request.

15. **title** : The loan title provided by the borrower
16. **dti** : A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower's self-reported monthly income.
17. **earliest_cr_line** :The month the borrower's earliest reported credit line was opened
18. **open_acc** : The number of open credit lines in the borrower's credit file.
19. **pub_rec** : Number of derogatory public records
20. **revol_bal** : Total credit revolving balance
21. **revol_util** : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
22. **total_acc** : The total number of credit lines currently in the borrower's credit file
23. **initial_list_status** : The initial listing status of the loan. Possible values are – W, F
24. **application_type** : Indicates whether the loan is an individual application or a joint application with two co-borrowers
25. **mort_acc** : Number of mortgage accounts.
26. **pub_rec_bankruptcies** : Number of public record bankruptcies
27. **Address**: Address of the individual

In [6]:
```
# Analysis of categorical features
df.describe(include = "object")

# The maximum count is 396030. However, some features have count less than that.
# need to look into that features specifically
# Maximum loan disburesed for 36 months period and maximum loan applicants have mor
# Most of the loans have been fully paid off
# Maximum loans have been disbursed for the purpose of debt consolidation
# Maximum application type is Individual
```

Out[6]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	i
count	396030	396030	396030	373103	377729	396030	396030	
unique	2	7	35	173105	11	6	3	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verified	
freq	302005	116018	26655	4389	126041	198348	139563	

In [7]:
```
# Statistical Analysis of numerical features
df.describe()

# The minimum and maximum loan amounts are 500 and 40000 respectively.
# The minimum and maximum interest rates are 5.32 and 30.99 respectively.
# The minimum and maximum intallments are 16 and 1533 respectively.
```

Out[7]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000

Duplicacy Check

In [8]:

```
# Check for duplicate values in dataset
df.duplicated().sum()

# There are no duplicate records in dataset.
```

Out[8]: 0

Missing values Check

In [9]:

```
# Check for all missing values in dataset
df.isnull().sum()

# Missing values are present in six features: "emp_title", "emp_length", "title", '
# "mort_acc", and "pub_rec_bankruptcies"
# For modeling, need to treat all missing values
```

Out[9]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

In [10]:
```
# Calculate the percentage of missing values in each feature.
df.isnull().sum()/len(df.index)*100

# "emp_title" constitutes of 5.78% of missing values
# "emp_length" contains 4.62% of missing values
# "title" contains 0.44% of missing values
# "revol_util" contains 0.07% of missing values
# "mort_acc" contains 9.54% of missing values
# "pub_rec_bankruptcies" contains 0.13% of missing values
```

Out[10]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

Treatment of all missing values - Imputation

In [11]:
```
# Features "emp_title", "emp_length", and "mort_acc" contains more than 1% of missi
# Hence, need to treat missing values in these features.
# As features "title", "revol_util", and "pub_rec_bankruptcies" contains less than
# The missing records for this features can be deleted.
```

In [12]:
```
# There are total 22927 values are missing in emp_title. It is huge number.
# Hence, instead of impute with anything, assign these missing values with new titl
df["emp_title"].fillna("unknown_job", inplace = True)
```

In [13]:
```
df["emp_length"].value_counts()
```

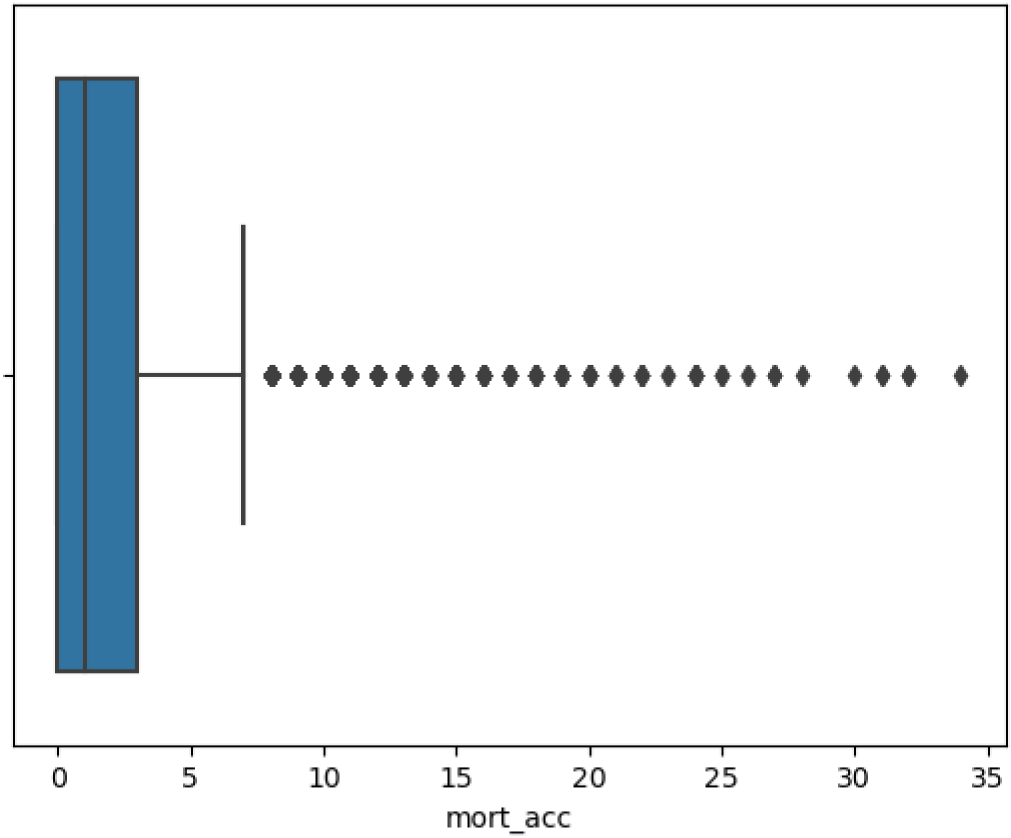
Out[13]:
```
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: emp_length, dtype: int64
```

In [14]:
```
df["emp_length"].value_counts().median()

# median falls between '1 year' and '4 years'.
# Therefore, best for median value imputation will be '2.5 years' but as it is floa
# round it to the nearest whole number, that is '3 years' as a reasonable imputatio
```

Out[14]:
```
25882.0
```

In [15]:
```
# There are total 18301 values are missing in emp_length.
# Assign these missing values with new length "unknown_years"
df["emp_length"].fillna("3 years", inplace = True)
```

In [16]:
```
# There are maxmium missing values in feature "mort_acc".
# "mort_acc" is a numerical feature. Plot boxplot to have more clarity for imputati
sns.boxplot(df["mort_acc"])
plt.show()

# There are many outliers in this feature. deep analysis is required for imputatior
```



In [17]:
```
# display(df["mort_acc"].value_counts())
print("--------Mean----------")
display(df["mort_acc"].mean())
print("--------Median----------")
display(df["mort_acc"].median())

# 0 have occured frequently but it would be wrong to assume that mortgage account c
# Mean is 1.81 and median is 1. It is better to impute the null values with median.
```

```
--------Mean----------
1.8139908160844138
--------Median----------
1.0
```

In [18]:
```
# Impute missing values of "mort_acc" feature with its median value
df['mort_acc'].fillna(df['mort_acc'].median(), inplace = True)
```

In [19]:
```
# delete NaN values from dataset.
df.dropna(inplace = True)
```

In [20]:
```
# Check again for all missing values
df.isnull().sum().sum()

# There are 0 missing values in dataset now
```

Out[20]: 0

In [21]:
```
df.head()
```

Out[21]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_o
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	M
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	M

5 rows × 27 columns

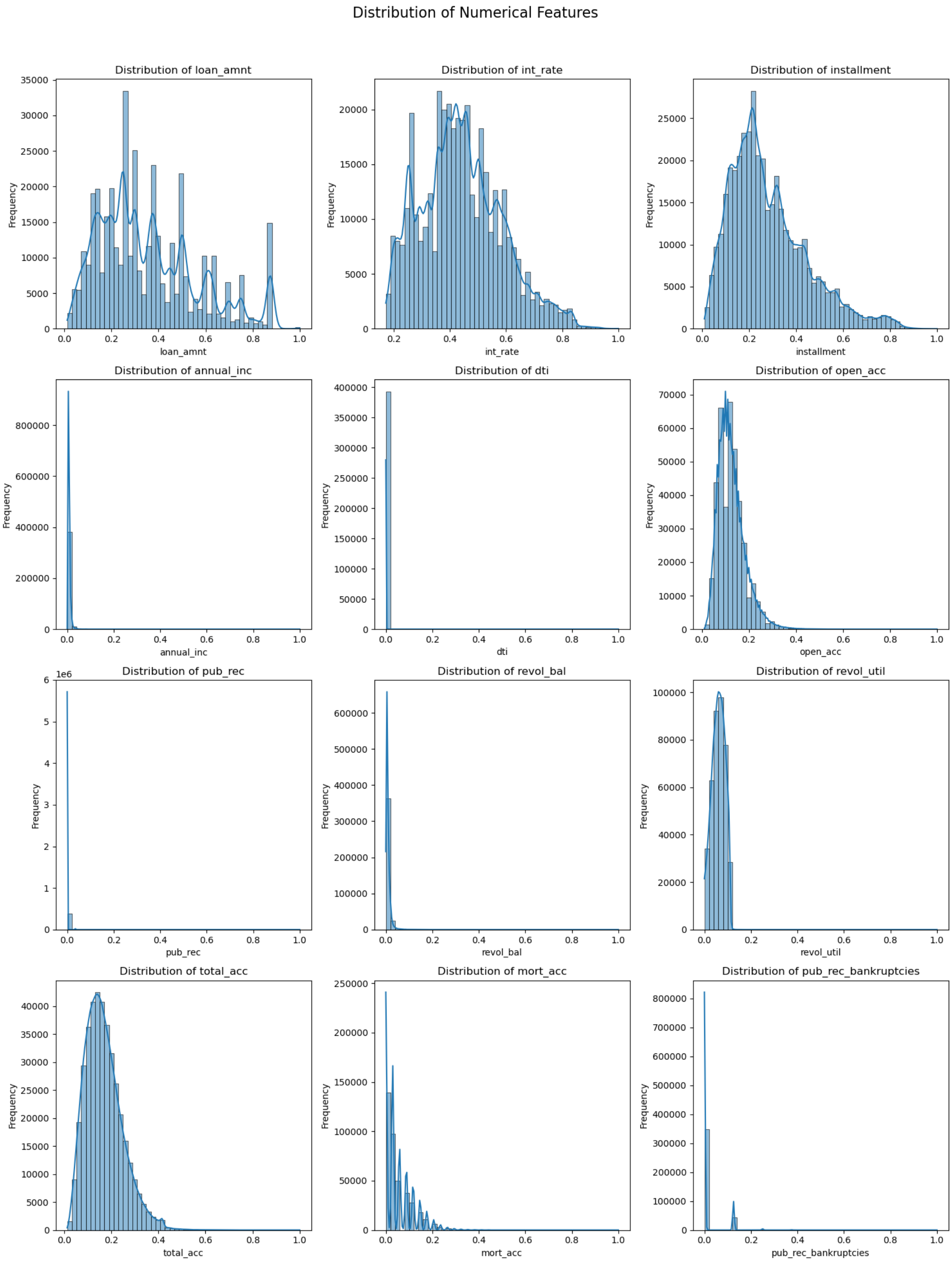
Univariate Analysis

In [22]:
```
# Make list of all numerical features
num_features = df.select_dtypes('float64').columns.tolist()
num_features
```

Out[22]:
```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'mort_acc',
 'pub_rec_bankruptcies']
```

In [23]:
```
# Set up the subplots
fig, axes = plt.subplots(4, 3, figsize=(15, 20))
fig.suptitle('Distribution of Numerical Features', fontsize=16)
# Flatten the axes array for easier indexing
axes = axes.flatten()
# Plot histograms for each numerical feature
for i, feature in enumerate(num_features):
    sns.histplot(df[feature] / df[feature].max(), kde=True, bins=50, ax=axes[i],pal
    # df[i].max()calculates the maximum value in the selected column and dividing t
    # maximum value scales the values between 0 and 1, hence normalizing the data.
    # that the histograms are comparable, especially if the numerical features have
    axes[i].set_title("Distribution of {}".format(feature))
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frequency")
```

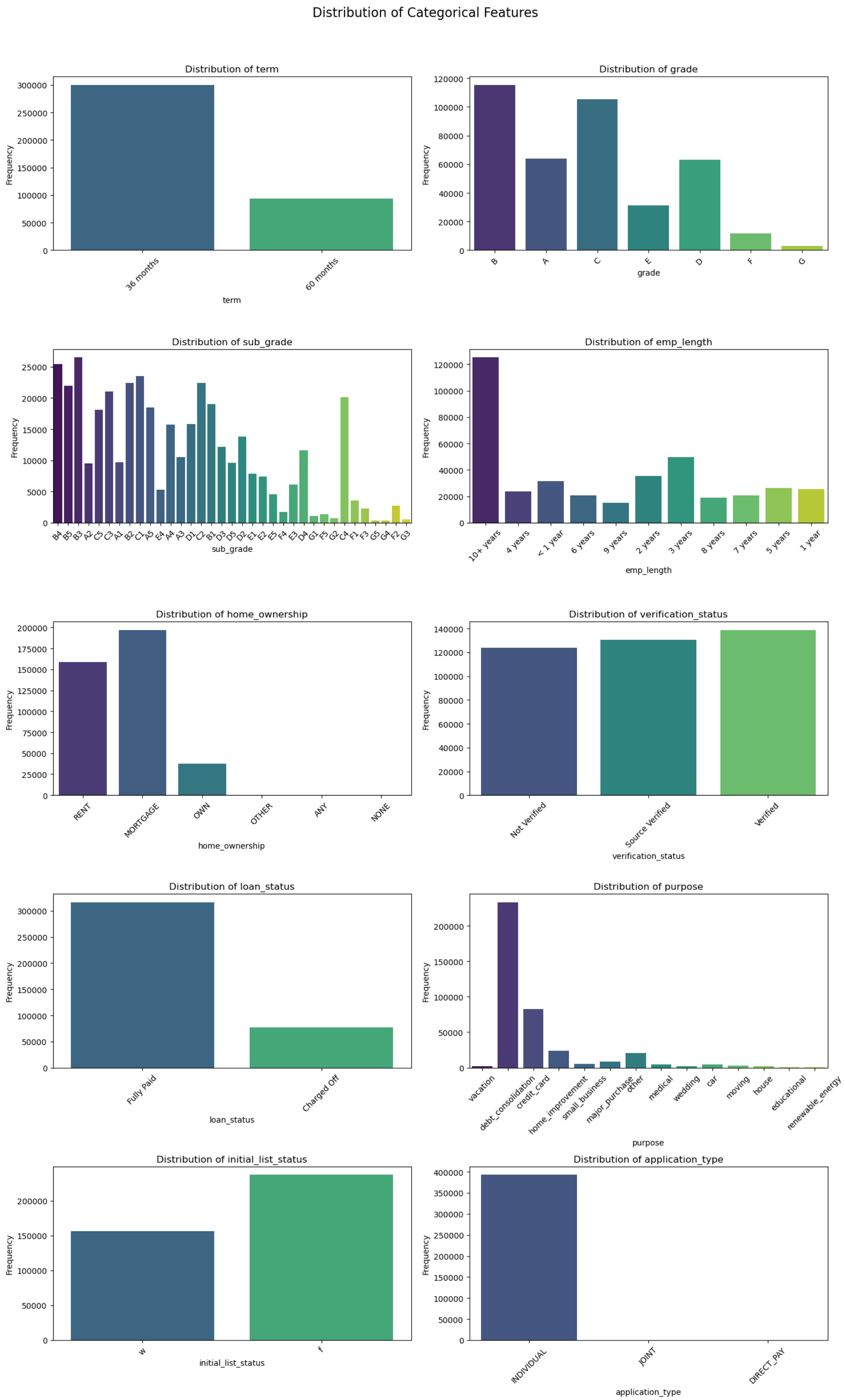
```python
# Adjust layout to prevent overlap of titles and labels
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



In [24]:
```python
# Make list of all categorical features
cat_features = ["term", "grade", "sub_grade", "emp_length", "home_ownership", "veri
                "purpose", "initial_list_status", "application_type"]
cat_features

# features such as "emp_title", "title", and "address" are object type but they are
# These features are not giving much relevance also.
# features like "issue_d" and  "earliest_cr_line" are datetime datatype.
# They do not provide much relevance in analyis separtely. However, their differenc
```

Out[24]:
```
['term',
 'grade',
 'sub_grade',
 'emp_length',
 'home_ownership',
 'verification_status',
 'loan_status',
 'purpose',
 'initial_list_status',
 'application_type']
```

In [25]:
```python
# Plot the graphs for categorical features.
# Set up the subplots
fig, axes = plt.subplots(5, 2, figsize=(15, 25))
fig.suptitle('Distribution of Categorical Features', fontsize=16)
# Flatten the axes array for easier indexing
axes = axes.flatten()
# Plot histograms for each categorical feature
for i, feature in enumerate(cat_features):
    sns.countplot(data = df, x = feature, ax=axes[i], palette='viridis')
    axes[i].set_title("Distribution of {}".format(feature))
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frequency")
    axes[i].tick_params(axis='x', rotation=45)
# Adjust layout to prevent overlap of titles and labels
plt.tight_layout(rect=[0, 0, 1, 0.96])
# Show the plot
plt.show()
```



In [26]:
```
len(df[df["loan_status"] == "Fully Paid"])/len(df)*100
```

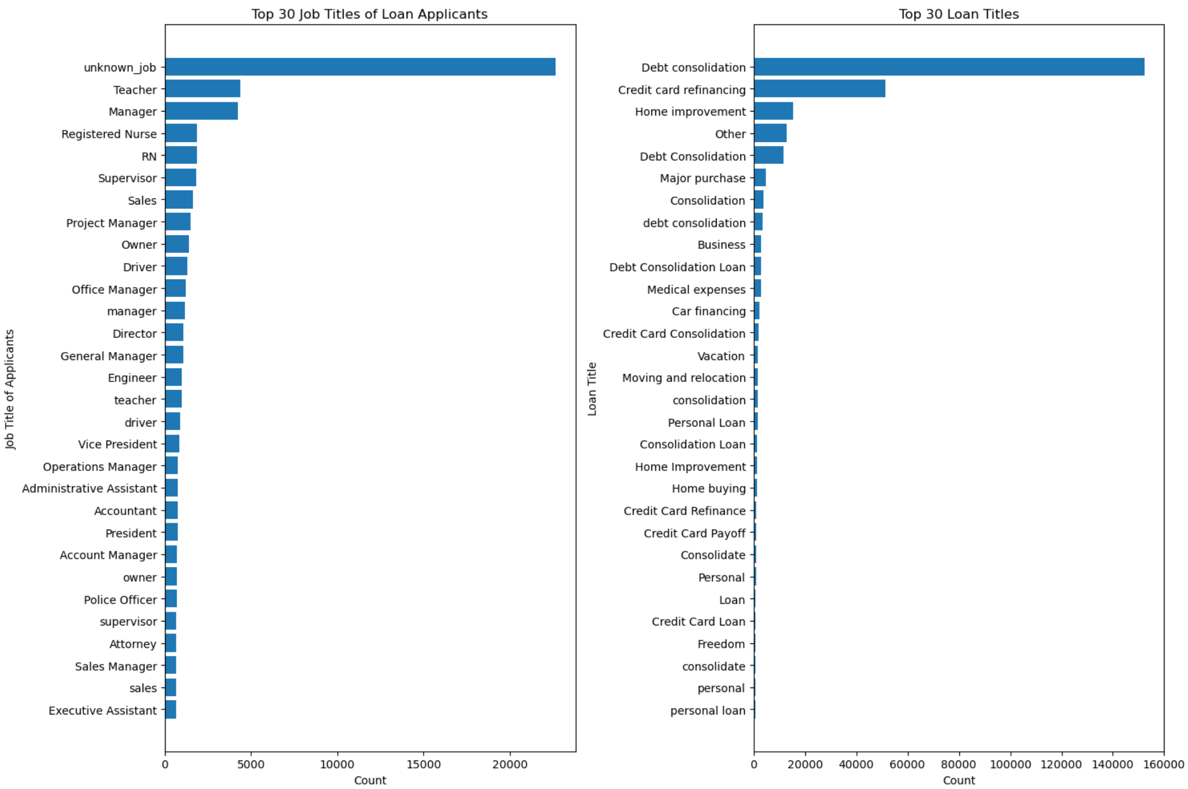
Out[26]: 80.38097416542767

1. What percentage of customers have fully paid their Loan Amount?

Ans: 80.38%

3. The majority of people have home ownership as MORTGRAGE.

In [27]:
```
# Analysis of feature "emp_title" and "title"
plt.figure(figsize=(15, 10))

# Subplot 1: emp_title
plt.subplot(1, 2, 1)
df_emp_title_counts = df['emp_title'].value_counts().nlargest(30)
df_emp_title_counts = df_emp_title_counts.sort_values(ascending=True)  # Sort in de
plt.barh(df_emp_title_counts.index, df_emp_title_counts)
plt.title("Top 30 Job Titles of Loan Applicants")
plt.xlabel("Count")
plt.ylabel("Job Title of Applicants")
plt.tight_layout()

# Subplot 2: title
plt.subplot(1, 2, 2)
df_title_counts = df['title'].value_counts().nlargest(30)
df_title_counts = df_title_counts.sort_values(ascending=True)  # Sort in decreasing
plt.barh(df_title_counts.index, df_title_counts)
plt.title("Top 30 Loan Titles")
plt.xlabel("Count")
plt.ylabel("Loan Title")
plt.tight_layout()

# Show the plots
plt.show()
```



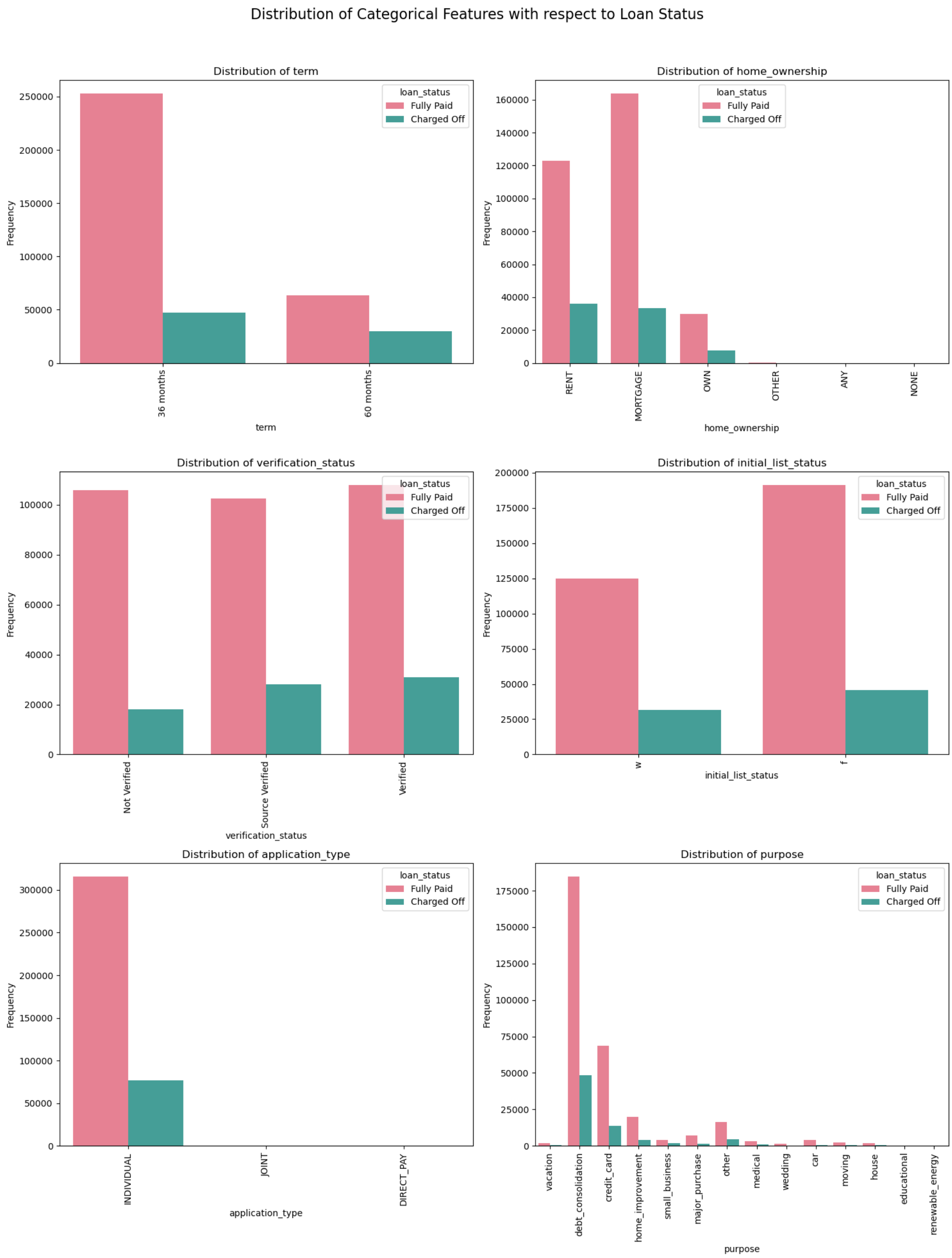
5. Name the top 2 afforded job titles.

Ans: Teacher and Manager

Bivariate Analysis

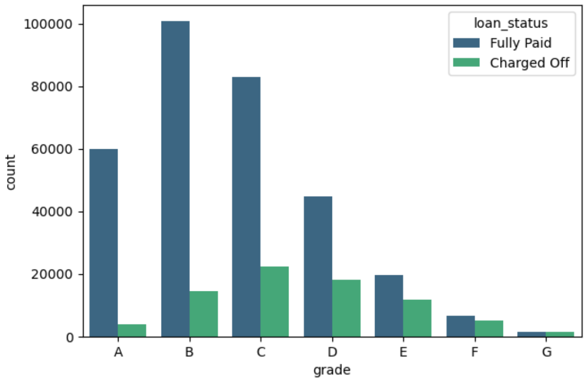
In [28]:
```
# Bivariate Analysis must be done to analyse the effect of each feature on loan sta
cat_features_1 = ["term", "home_ownership", "verification_status",
                  "initial_list_status", "application_type", "purpose"]
```

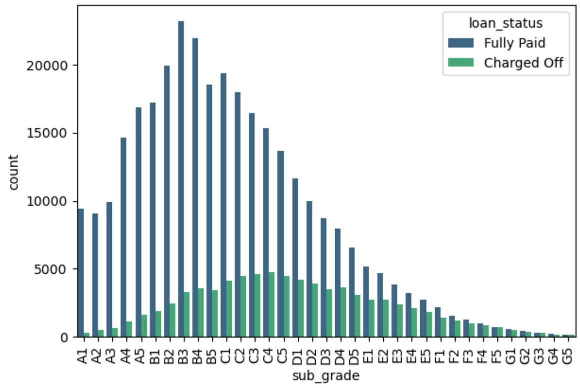
In [29]:
```
# Analyse variation of features lsited in "cat_features_1" with respect to loan_sta
# Set up the subplots
fig, axes = plt.subplots(3, 2, figsize=(15, 20))
fig.suptitle('Distribution of Categorical Features with respect to Loan Status', fo
# Flatten the axes array for easier indexing
axes = axes.flatten()
# Plot histograms for each categorical feature
for i, feature in enumerate(cat_features_1):
    sns.countplot(data = df, x = feature, ax=axes[i], hue='loan_status', palette='h
    axes[i].set_title("Distribution of {}".format(feature))
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frequency")
    axes[i].tick_params(axis='x', rotation=90)
# Adjust layout to prevent overlap of titles and labels
plt.tight_layout(rect=[0, 0, 1, 0.96])
# Show the plot
plt.show()
```



In [30]:

```
# Analysis of "grade" and "sub_grade" features with respect to loan_status
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
grade = sorted(df.grade.unique().tolist())
sns.countplot(x='grade', data=df, hue='loan_status', order=grade, palette='viridis'

plt.subplot(2, 2, 2)
sub_grade = sorted(df.sub_grade.unique().tolist())
g = sns.countplot(x='sub_grade', data=df, hue='loan_status', order=sub_grade, palet
g.set_xticklabels(g.get_xticklabels(), rotation=90)

plt.show()
```

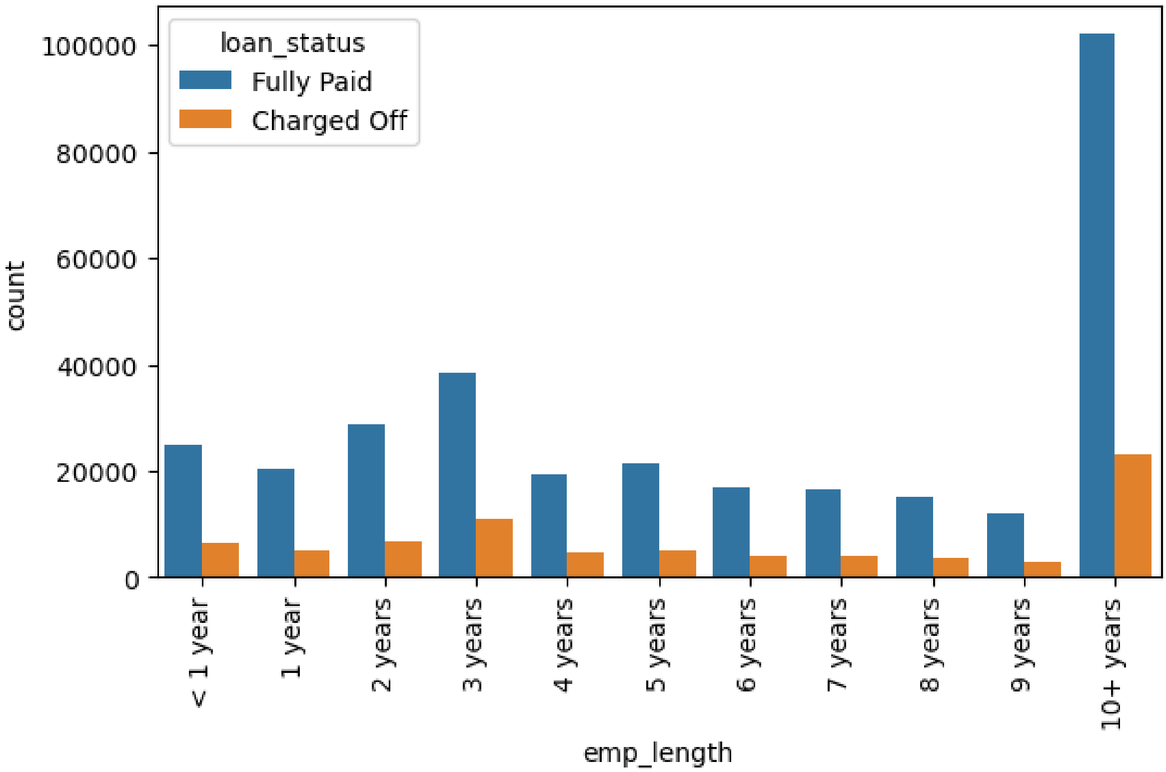




4. People with grades 'A' are more likely to fully pay their loan. (T/F)

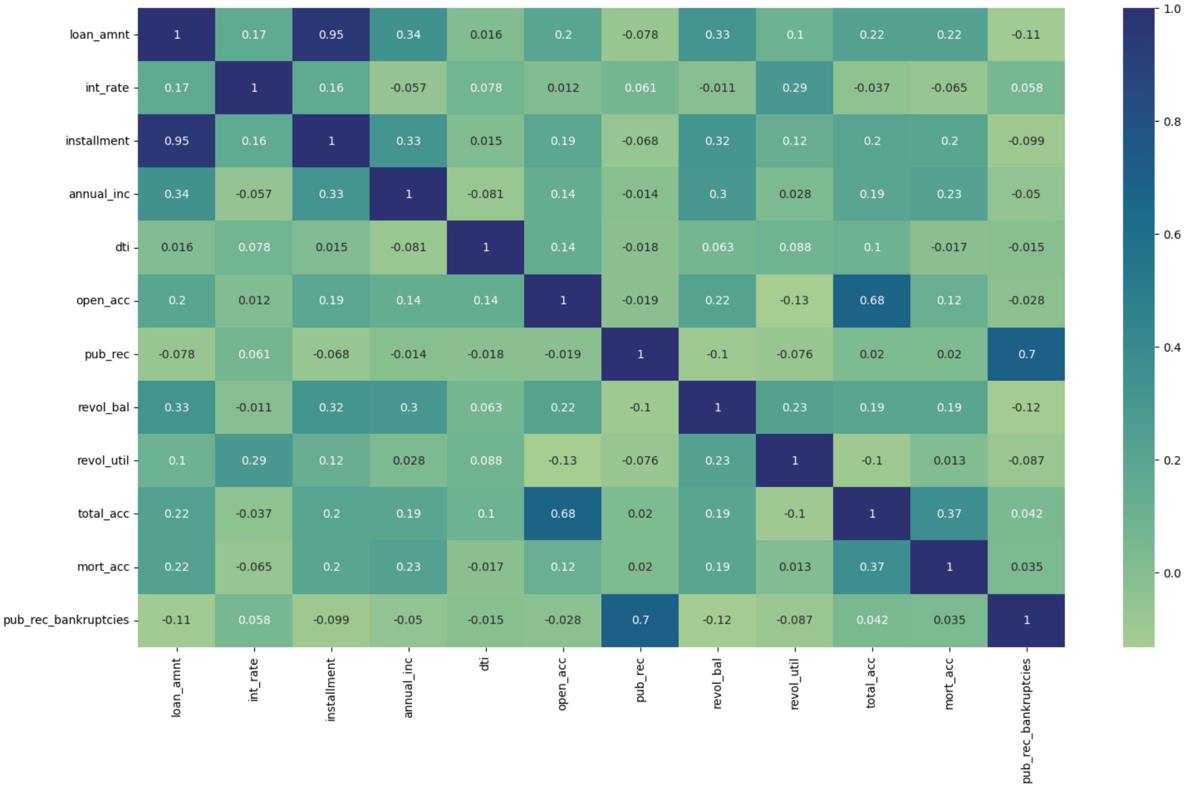
Ans: YES

In [31]:
```
# Analysis of "emp_length" feature with respect to loan_status
plt.figure(figsize=(7,4))

order = ['< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years',
         '6 years', '7 years', '8 years', '9 years', '10+ years',]
g=sns.countplot(x='emp_length',data=df,hue='loan_status',order=order)
g.set_xticklabels(g.get_xticklabels(),rotation=90)
plt.show()
```



Analysis of correlation of Numerical features

In [32]:
```
# Plot heatmap for correlation of all numerical features.
plt.figure(figsize=(18,10))
sns.heatmap(df.corr(), cmap = 'crest', annot = True)
plt.show()

# Maximum or high correlation can be seen between feature "installment" and "loan_a
# Correlation can also be seen between feature "pub_rec" and "pub_rec_bankruptcies'
# And between feature "open_acc" and "total_acc".
```



2. Comment about the correlation between Loan Amount and Installment features.

Ans: Correlation between Loan Amount and Installment features is 0.95.

Data Preprocessing

Feature Engineering

In [33]:
```
def flag(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
    else:
        return number
```

In [34]:
```
df['pub_rec']=df['pub_rec'].apply(flag)
df['mort_acc']=df['mort_acc'].apply(flag)
df['pub_rec_bankruptcies']=df['pub_rec_bankruptcies'].apply(flag)
```

In [35]:
```
display(df['pub_rec'].value_counts())
display(df['mort_acc'].value_counts())
display(df['pub_rec_bankruptcies'].value_counts())
```

```
0    336074
1     57391
Name: pub_rec, dtype: int64
1    254411
0    139054
Name: mort_acc, dtype: int64
0    348599
1     44866
Name: pub_rec_bankruptcies, dtype: int64
```

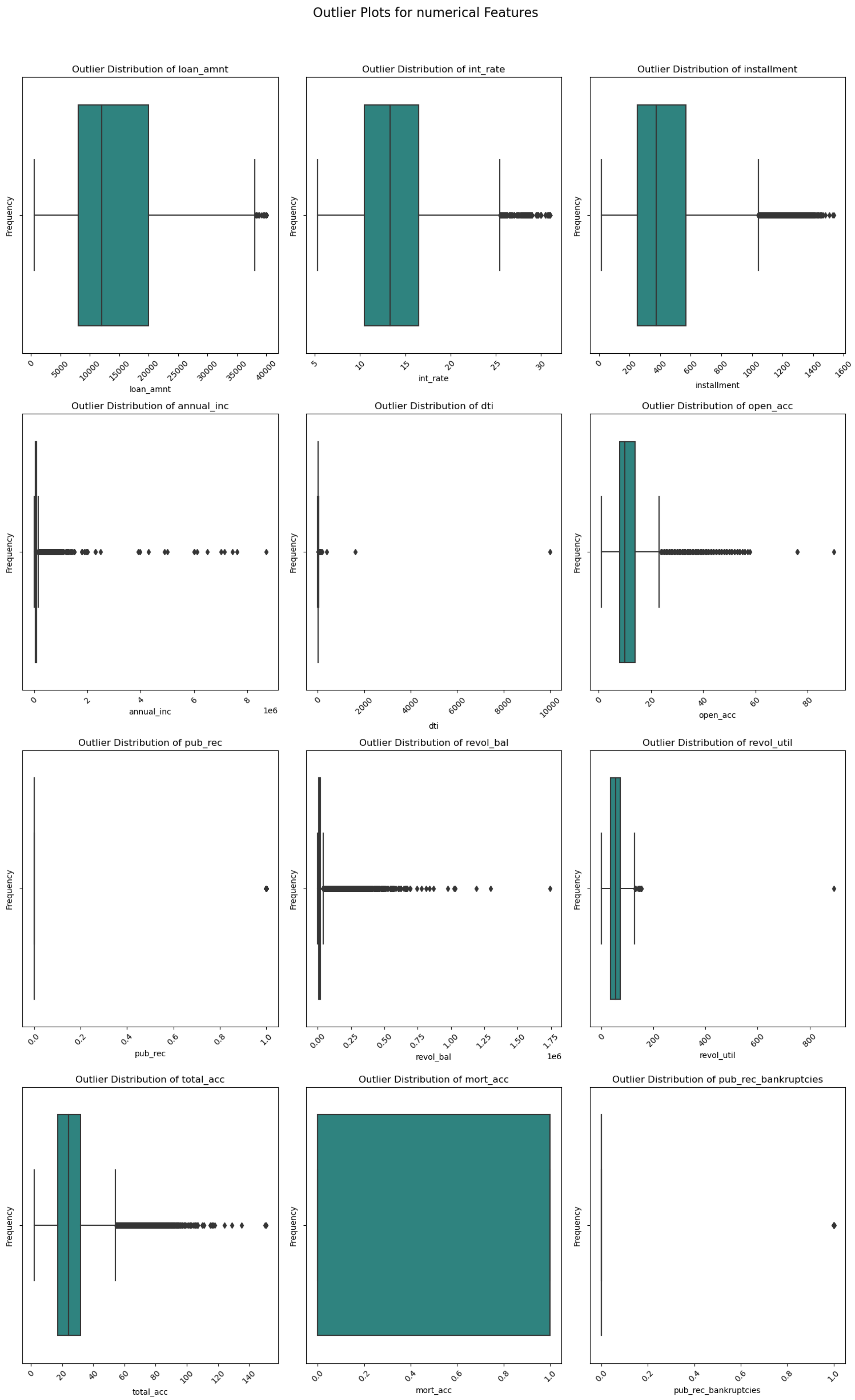
Outlier Detection

In [36]:

```
def outlier_plot(i):
    plt.figure(figsize=(8,5))
    sns.boxplot(x=df[i])
    plt.title('Boxplot for {}'.format(i))
    plt.show()
```

In [37]:

```
# Plot the graphs for outliers in numerical features.
# Set up the subplots
fig, axes = plt.subplots(4, 3, figsize=(15, 25))
fig.suptitle('Outlier Plots for numerical Features', fontsize=16)
# Flatten the axes array for easier indexing
axes = axes.flatten()
# Plot boxplots for each numerical feature
for i, feature in enumerate(num_features):
    sns.boxplot(data = df, x = feature, ax=axes[i], palette='viridis')
    axes[i].set_title("Outlier Distribution of {}".format(feature))
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frequency")
    axes[i].tick_params(axis='x', rotation=45)
# Adjust layout to prevent overlap of titles and labels
plt.tight_layout(rect=[0, 0, 1, 0.96])
# Show the plot
plt.show()

# As flags has been set for three features 'pub_rec','mort_acc' and 'pub_rec_bankru
# No treatment of outliers in these features is necessary
```

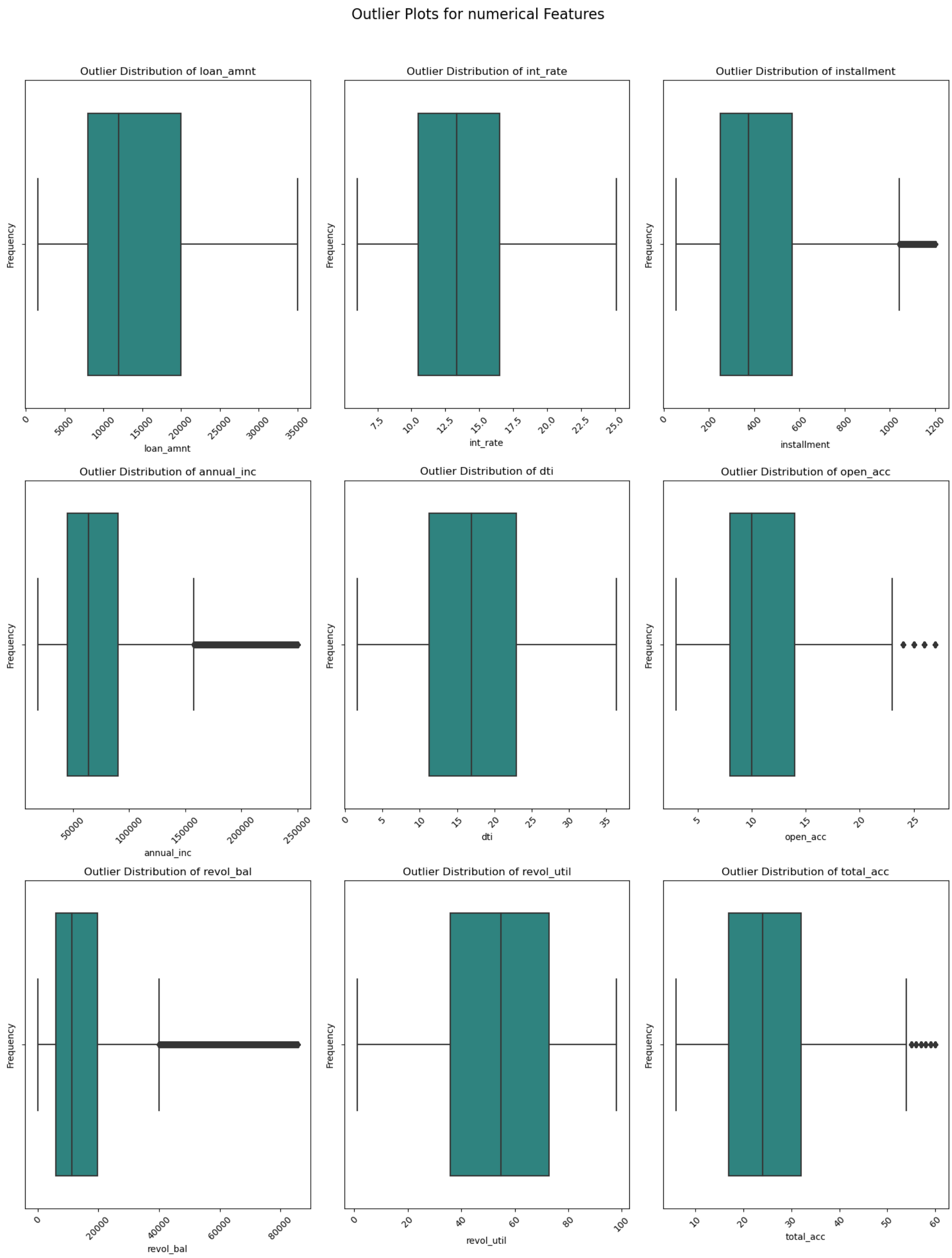


In [38]:
```
num_feat = ['loan_amnt','int_rate','installment','annual_inc','dti','open_acc','rev
```

Outlier Treatment

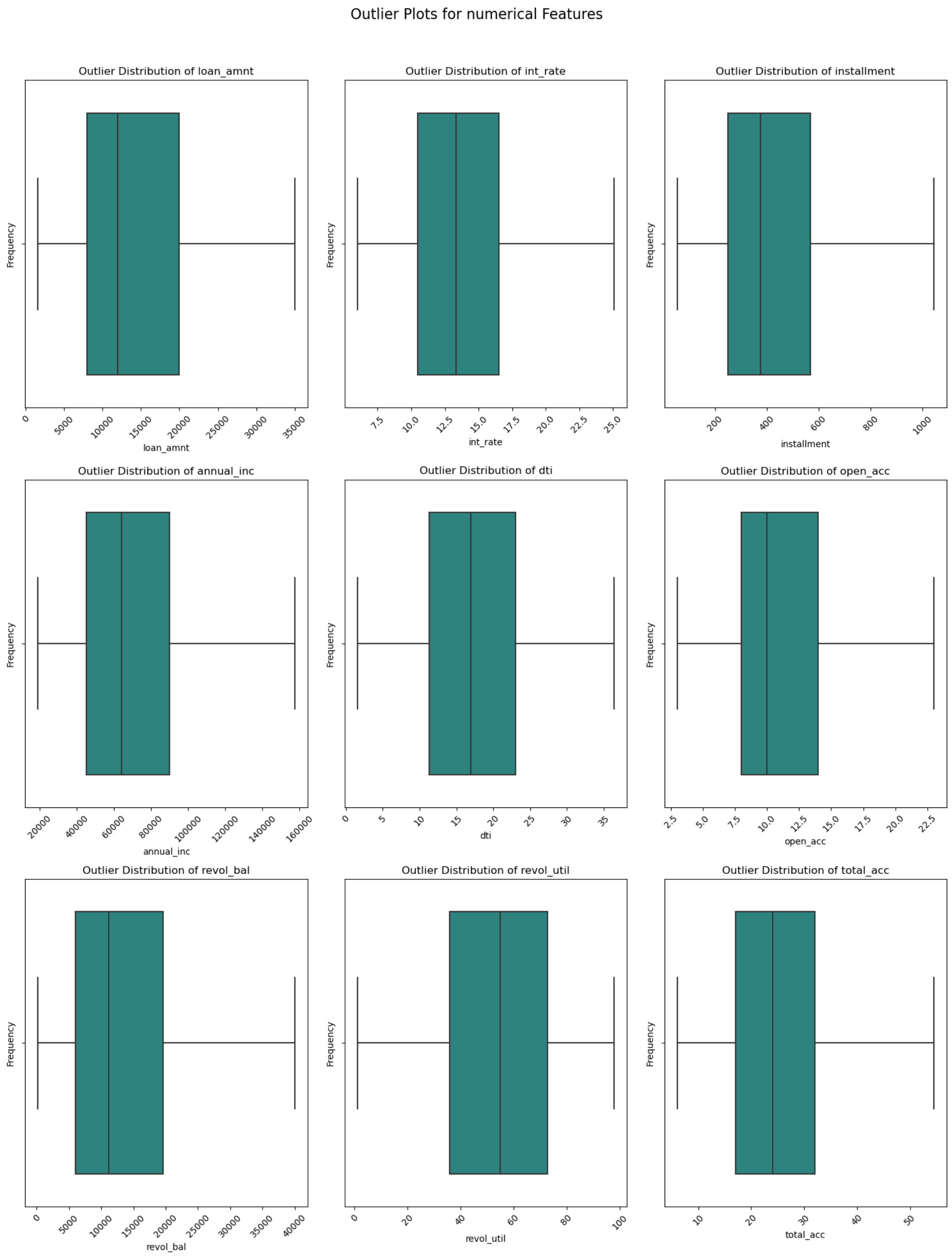
In [39]:
```
# For treatment of outliers, use percentile capping method.
# Set values above the 99th percentile to the value at the 99th percentile
# and values below the 1th percentile to the value at the 1th percentile.
for col in num_feat:
    percentiles = df[col].quantile([0.01, 0.99]).values
    df[col] = np.clip(df[col], percentiles[0], percentiles[1])
```

In [40]:
```
# Plot the graphs for outliers in numerical features.
# Set up the subplots
fig, axes = plt.subplots(3, 3, figsize=(15, 20))
fig.suptitle('Outlier Plots for numerical Features', fontsize=16)
# Flatten the axes array for easier indexing
axes = axes.flatten()
# Plot boxplots for each numerical feature
for i, feature in enumerate(num_feat):
    sns.boxplot(data = df, x = feature, ax=axes[i], palette='viridis')
    axes[i].set_title("Outlier Distribution of {}".format(feature))
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frequency")
    axes[i].tick_params(axis='x', rotation=45)
# Adjust layout to prevent overlap of titles and labels
plt.tight_layout(rect=[0, 0, 1, 0.96])
# Show the plot
plt.show()

# Still outliers can be shown in some features like "installment", "annual_inc", "c
# "revol_bal", "total_acc", "mort_acc", and "pub_rec_bankruptcies"
```



In [41]:
```
# For remaining outliers, use IQR method to remove the outliers
for col in num_feat:
    #Calculate IQR
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #Identify Potential Outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Treatment of Outliers
    # Replace outliers with the lower/upper bound, or remove them, based on your pr
    df[col] = df[col].clip(lower=lower_bound, upper=upper_bound)
```

In [42]:
```
# Check again for outliers
# Plot the graphs for outliers in numerical features.
# Set up the subplots
fig, axes = plt.subplots(3, 3, figsize=(15, 20))
fig.suptitle('Outlier Plots for numerical Features', fontsize=16)
# Flatten the axes array for easier indexing
axes = axes.flatten()
# Plot boxplots for each numerical feature
for i, feature in enumerate(num_feat):
    sns.boxplot(data = df, x = feature, ax=axes[i], palette='viridis')
    axes[i].set_title("Outlier Distribution of {}".format(feature))
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frequency")
    axes[i].tick_params(axis='x', rotation=45)
# Adjust layout to prevent overlap of titles and labels
plt.tight_layout(rect=[0, 0, 1, 0.96])
# Show the plot
plt.show()
```



Data Cleaning

In [43]:
```
# Analyse one by one all the features
# Start by least relevant feature
df.columns
```

Out[43]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

1. Map the target variable "loan_status" as 0 and 1.
2. "emp_title" feature has 172227 unique titles. This feature does not provide much relevance in building model. Hence, drop this feature.
3. "verification_status" feature actually conveys whether LoanTap has verified income or not. This may or may not be an important feature, hence, analysis must be done to ensure.
4. "purpose" is category mentioned by borrower. This can also be analysed.
5. "title" is loan title mentioned by borrower. There are 48472 titles are there and do not contribute much. Hence, drop this feature.
6. As earlier observed, "loan_amnt" and "installment" features shows high correlation, hence, one feature can be dropped from the dataset
7. features like "term" and "emp_length" have object datatype. Convert them in numeric values by removing "months" from "term" feature and "year/years", "+", & "<" from "emp_length" feature
8. features such as "issue_d" and "earliest_cr_line" are object datatype. Convert them in datetime format. And calculate the loan age by subtracting issue date from current date and credit line age by subtracting earliest credit line date from current date.
9. "Address" feature soesn't seem to be relevant. However, check whether any correlation exists between the zipcode (mentioned in address) and loan_status. If correlation does not exists, drop that feature.

In [44]:
```
# Mapping the target variable
df['loan_status']=df['loan_status'].map({'Fully Paid':0, 'Charged Off':1})
```

In [45]:
```
# Preprocessing of "emp_length"
# Split the numerical part and year/years part
df['emp_length'] = df['emp_length'].replace ( ['< 1 year'],'0 year')
df['emp_length'] = df['emp_length'].replace ( ['10+ years'],'10 years')
df[['emp_tenure_in_years','years']] = df['emp_length'].str.split(' ',expand=True)
df['emp_tenure_in_years']=df['emp_tenure_in_years'].astype(int)
```

In [46]:
```
# Preprocessing of "term"
df[['index','term_in_months','months']] = df['term'].str.split(' ',expand=True)
df["term_in_months"] = df["term_in_months"].astype(int)
```

In [47]:
```
# preprocessing of "issue_d" and "earliest_cr_line"
df['issue_d']=df['issue_d'].astype('datetime64[ns]')
df['earliest_cr_line']=df['earliest_cr_line'].astype('datetime64[ns]')
```

In [48]:
```
# Calculate loan_age and credit_line_age
import datetime as dt
df['current_date'] = pd.to_datetime(dt.date.today())
df['current_date']=df['current_date'].astype('datetime64[ns]')
df["credit_line_age"] = (df['current_date'] - df['earliest_cr_line']) / np.timedelt
df["loan_age"] = (df['current_date'] - df['issue_d']) / np.timedelta64(1, 'D')
```

In [49]:
```
# Create new column zipcode from address
df['zip_code'] = df["address"].apply(lambda x: x[-5:])
df['zip_code'] = df['zip_code'].astype('int')
```

Check association of newly added features with "loan_status"

In [50]:
```
# Check association of newly added features with loan_status using point Biserial
from scipy.stats import pointbiserialr
# The point-biserial correlation is used to measure the strength and direction of t
# a binary categorical variable and a continuous numerical variable.
def calculate_point_Biserial(col):
    # Calculate point-biserial correlation
    point_biserial_corr, p_value = pointbiserialr(df['loan_status'], col)
    # Print the results
    print("Point-Biserial Correlation Coefficient:", point_biserial_corr)
    print("P-value:", p_value)
    print("---------------------------------------------------------------")

# point-biserial correlation coefficient ranges from -1 to 1, where -1 indicates a
# relationship, 1 indicates a perfect positive relationship, and 0 indicates no rel
# The p-value assess the statistical significance of the correlation.
```

In [51]:
```
calculate_point_Biserial(df["loan_age"])
calculate_point_Biserial(df["credit_line_age"])
calculate_point_Biserial(df["zip_code"])
calculate_point_Biserial(df["emp_tenure_in_years"])
calculate_point_Biserial(df["term_in_months"])
```

```
Point-Biserial Correlation Coefficient: -0.059492718725678184
P-value: 2.457316619867536e-305
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.03878146019016709
P-value: 8.267949576599197e-131
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.3469725424215233
P-value: 0.0
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.020688488590169467
P-value: 1.6187000576120288e-38
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.17405072457438492
P-value: 0.0
---------------------------------------------------------------
```

In [52]:
```
# As point-biserial correlation coefficients of only "zip_code" and "term_in_months
# Other features do not add much relevance in modeling.
# Better to drop these columns.
```

In [53]:
```
# Drop "emp_title" and "title" from dataset
df.drop(['emp_title','title'],axis='columns',inplace=True)
```

In [54]:
```
# drop irrelevant columns
df.drop(['emp_length','years','term','months','index','current_date','earliest_cr_l
         "loan_age", "credit_line_age","emp_tenure_in_years"],axis='columns',inplac
```

In [55]:
```
# Mapping other variables
df["verification_status"]=df["verification_status"].map({'Verified':0,'Source Verif
df["grade"] = df["grade"].map({'A': 1, 'B':2, 'C':3,'D':4,'E':5,'F':6,'G':7})
df["initial_list_status"] = df["initial_list_status"].map({'w':0,'f':1})
df["application_type"] = df["application_type"].map({'INDIVIDUAL':1,'JOINT':2,'DIRE
df["home_ownership"] = df["home_ownership"].map({'MORTGAGE':1,'RENT':2,'OWN':3,'OTH
df["purpose"]=df["purpose"].map({'debt_consolidation':1,'credit_card':2,'home_impro
                                 'major_purchase':5,'small_business':6,'car':7,'med
                                 'vacation':10,'house':11,'wedding':12,'renewable_e
df["sub_grade"] = df["sub_grade"].map({'A1':1,'A2':2,'A3':3,'A4':4,'A5':5,'B1':6,'E
                                       'B5':10,'C1':11,'C2':12,'C3':13,'C4':14,'C5'
                                       'D3':18,'D4':19,'D5':20,'E1':21,'E2':22,'E3'
```

```
                                       'F1':26,'F2':27,'F3':28,'F4':29,'F5':30,'G1'
                                       'G4':34,'G5':35})
```

In [56]:
```
calculate_point_Biserial(df["verification_status"])
calculate_point_Biserial(df["grade"])
calculate_point_Biserial(df["initial_list_status"])
calculate_point_Biserial(df["application_type"])
calculate_point_Biserial(df["home_ownership"])
calculate_point_Biserial(df["purpose"])
calculate_point_Biserial(df["sub_grade"])
```

```
Point-Biserial Correlation Coefficient: -0.07791371091039892
P-value: 0.0
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.25743910994974234
P-value: 0.0
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.009959945049769672
P-value: 4.16570682024538e-10
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.005249939648697237
P-value: 0.0009907838023667215
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.054484974939249115
P-value: 2.2601404506525032e-256
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.008870495832257397
P-value: 2.6325367477761652e-08
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.2631360649722843
P-value: 0.0
---------------------------------------------------------------
```

In [57]:
```
# As point-biserial correlation coefficients of only "grade" and "sub_grade" are cc
# Other features do not add much relevance in modeling.
# Better to drop these columns.
# drop irrelevant columns
df.drop(['initial_list_status','application_type','home_ownership',
         "purpose"],axis='columns',inplace=True)
```

In [58]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 393465 entries, 0 to 396029
Data columns (total 18 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             393465 non-null  float64
 1   int_rate              393465 non-null  float64
 2   installment           393465 non-null  float64
 3   grade                 393465 non-null  int64  
 4   sub_grade             393465 non-null  int64  
 5   annual_inc            393465 non-null  float64
 6   verification_status   393465 non-null  int64  
 7   loan_status           393465 non-null  int64  
 8   dti                   393465 non-null  float64
 9   open_acc              393465 non-null  float64
 10  pub_rec               393465 non-null  int64  
 11  revol_bal             393465 non-null  float64
 12  revol_util            393465 non-null  float64
 13  total_acc             393465 non-null  float64
 14  mort_acc              393465 non-null  int64  
 15  pub_rec_bankruptcies  393465 non-null  int64  
 16  term_in_months        393465 non-null  int32  
 17  zip_code              393465 non-null  int32  
dtypes: float64(9), int32(2), int64(7)
memory usage: 54.0 MB
```

In [59]:
```
# Till now, features like grade, sub_grade,term_in_months, and zip_code appears to
# Analyse numerical features
calculate_point_Biserial(df["loan_amnt"])
calculate_point_Biserial(df["int_rate"])
calculate_point_Biserial(df["installment"])
calculate_point_Biserial(df["annual_inc"])
calculate_point_Biserial(df["dti"])
calculate_point_Biserial(df["open_acc"])
```

```
Point-Biserial Correlation Coefficient: 0.060183098429249016
P-value: 1.99956892708e-312
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.2482910332873778
P-value: 0.0
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.04232164333488121
P-value: 2.0311601019019225e-155
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.08225245281008454
P-value: 0.0
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.13215605663275865
P-value: 0.0
---------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.02783978922657529
P-value: 2.5857950464317596e-68
---------------------------------------------------------------
```

In [60]:
```
calculate_point_Biserial(df["revol_bal"])
calculate_point_Biserial(df["revol_util"])
calculate_point_Biserial(df["total_acc"])
calculate_point_Biserial(df["mort_acc"])
calculate_point_Biserial(df["pub_rec"])
calculate_point_Biserial(df["pub_rec_bankruptcies"])
```

```
Point-Biserial Correlation Coefficient: -0.002595847984993309
P-value: 0.10346348367254961
----------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.082026193588034
P-value: 0.0
----------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.018604828780407607
P-value: 1.7891442938525038e-31
----------------------------------------------------------------
Point-Biserial Correlation Coefficient: -0.07605831145294961
P-value: 0.0
----------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.01823070915556885
P-value: 2.7489777820886292e-30
----------------------------------------------------------------
Point-Biserial Correlation Coefficient: 0.008451439945327117
P-value: 1.1492133354641306e-07
----------------------------------------------------------------
```

In [61]:

```
# As point-biserial correlation coefficients of only "int_rate" and "dti" are consi
# Other features do not add much relevance in modeling.
# Better to drop these columns.
# drop irrelevant columns
df.drop(["installment", "open_acc","revol_bal","total_acc",
        "pub_rec", "pub_rec_bankruptcies"],axis='columns',inplace=True)
```

In [62]:

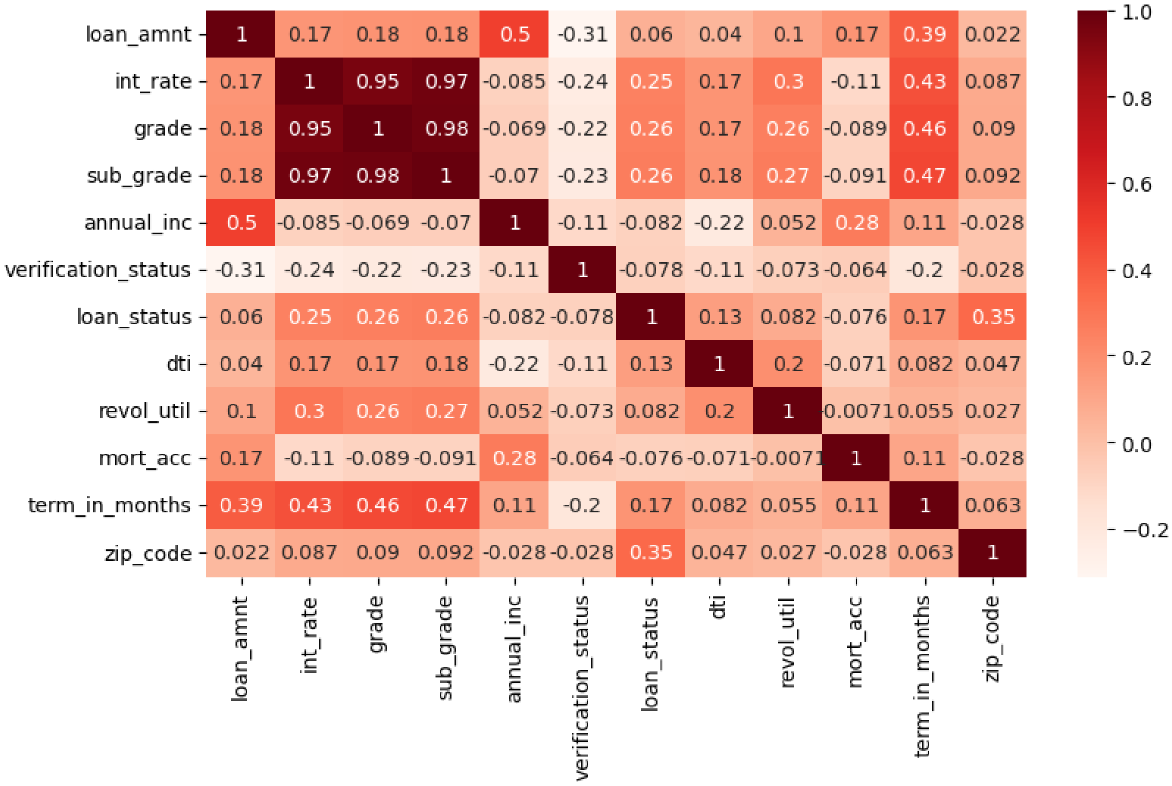
```
df.head()
# Dataset is now prepared for modeling
```

Out[62]:

	loan_amnt	int_rate	grade	sub_grade	annual_inc	verification_status	loan_status	dti	revol_
0	10000.0	11.44	2	9	117000.0	2	0	26.24	
1	8000.0	11.99	2	10	65000.0	2	0	22.05	
2	15600.0	10.49	2	8	43057.0	1	0	12.79	
3	7200.0	6.49	1	2	54000.0	2	0	2.60	
4	24375.0	17.27	3	15	55000.0	0	1	33.95	

In [63]:

```
plt.figure(figsize=(9,5))
sns.heatmap(df.corr(),annot=True,cmap='Reds')
plt.show()
```



In [64]:

```
df.corr()
# Based on correlation matrix values:
# 1. "loan_amnt" have correlation of 0.497813 with "annual_inc", 0.393746 with "ter
#     negative correlation of 0.311559 with "verification_status".
# 2. "int_rate" have high collinearity with "grade","sub_grade" and "term_in_months
#     and 0.434191 respectively.
# 3. "term_in_months" have high collinearity with "loan_amnt","int_rate","grade" ar
#     0.393746,0.434191,0.457997 and 0.468760 respectively.
# 5. "grade" and "sub_grade" have high collinearity of 0.977553
# As "int_rate" have high collinearity with "grade","sub_grade" and "term_in_months
# multicollinearity. Therefore, drop this feature "int_rate".
# Based on same concept, drop "term_in_months", "grade" and "loan_amnt" also.
```

Out[64]:

	loan_amnt	int_rate	grade	sub_grade	annual_inc	verification_status	loar
loan_amnt	1.000000	0.168085	0.175249	0.181985	0.497813	-0.311559	0
int_rate	0.168085	1.000000	0.952447	0.973957	-0.085292	-0.235167	0
grade	0.175249	0.952447	1.000000	0.977553	-0.069055	-0.219437	0
sub_grade	0.181985	0.973957	0.977553	1.000000	-0.070498	-0.229422	0
annual_inc	0.497813	-0.085292	-0.069055	-0.070498	1.000000	-0.114378	-0
verification_status	-0.311559	-0.235167	-0.219437	-0.229422	-0.114378	1.000000	-0
loan_status	0.060183	0.248291	0.257439	0.263136	-0.082252	-0.077914	1
dti	0.040053	0.174079	0.171101	0.176071	-0.219688	-0.114540	0
revol_util	0.100411	0.295655	0.259202	0.269532	0.051630	-0.072513	0
mort_acc	0.172848	-0.112321	-0.088589	-0.090588	0.275407	-0.063886	-0
term_in_months	0.393746	0.434191	0.457997	0.468760	0.107603	-0.196441	0
zip_code	0.022081	0.087037	0.090240	0.092339	-0.027716	-0.027568	0

In [65]:
```python
df.drop(["int_rate","grade", "loan_amnt","term_in_months"],axis='columns',inplace=T
```

In [66]:
```python
df.corr()
```

Out[66]:

	sub_grade	annual_inc	verification_status	loan_status	dti	revol_util	mo
sub_grade	1.000000	-0.070498	-0.229422	0.263136	0.176071	0.269532	-0.(
annual_inc	-0.070498	1.000000	-0.114378	-0.082252	-0.219688	0.051630	0.2
verification_status	-0.229422	-0.114378	1.000000	-0.077914	-0.114540	-0.072513	-0.(
loan_status	0.263136	-0.082252	-0.077914	1.000000	0.132156	0.082026	-0.(
dti	0.176071	-0.219688	-0.114540	0.132156	1.000000	0.195059	-0.(
revol_util	0.269532	0.051630	-0.072513	0.082026	0.195059	1.000000	-0.(
mort_acc	-0.090588	0.275407	-0.063886	-0.076058	-0.070825	-0.007131	1.(
zip_code	0.092339	-0.027716	-0.027568	0.346973	0.046787	0.027097	-0.(

8. Which were the features that heavily affected the outcome?

Ans: *loan_status feature is target variable and hence conclude as outcome. This feature has maximum correlation with zip_code of* **0.346973.**

The correlation coefficient of 0.346973 suggests a moderate positive linear relationship between the target and the "zip_code" variable.As the feature increases, there is a tendency for the "zip_code" to increase moderately.This correlation could imply that the "zip_code" variable contains some information about the feature, or vice versa.

9. Will the results be affected by geographical location? (Yes/No)

Ans: ***YES****, there is probability that results may affect by geographical location. As geographical location based on zip code and zip_code feature has moderate positive linear relationship with result.*

In [67]:
```python
df1 = df
```

In [68]:
```python
df1
```

Out[68]:

	sub_grade	annual_inc	verification_status	loan_status	dti	revol_util	mort_acc	zip_coc
0	9	117000.0	2	0	26.24	41.8	0	2269
1	10	65000.0	2	0	22.05	53.3	1	511
2	8	43057.0	1	0	12.79	92.2	0	511
3	2	54000.0	2	0	2.60	21.5	0	81
4	15	55000.0	0	1	33.95	69.8	1	1165
...	...	...	...	...	...	...	...	
396025	9	40000.0	1	0	15.63	34.3	0	3072
396026	11	110000.0	1	0	21.45	95.7	1	511
396027	6	56500.0	0	0	17.56	66.9	0	7046
396028	12	64000.0	0	0	15.88	53.8	1	2959
396029	12	42996.0	0	0	8.32	91.3	1	4805

393465 rows × 8 columns

Data modeling

In [69]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import roc_auc_score,roc_curve, auc
from sklearn.metrics import precision_recall_curve, average_precision_score
```

In [70]:
```
# Assign labels and target vector
X = df1.drop('loan_status',axis=1)
y = df1["loan_status"]
```

Scaling- MinMax

In [71]:
```
scaler = MinMaxScaler()
x= scaler.fit_transform(X)
```

In [72]:
```
x_train, x_test, y_train, y_test =train_test_split(x,y,test_size=0.20,stratify=y,ra
```

Data Modeling

In [73]:
```
lr=LogisticRegression(max_iter=1000)
lr.fit(x_train,y_train)
```

Out[73]: LogisticRegression(max_iter=1000)

In [74]:
```
y_pred = lr.predict(x_test)
print('Accuracy of Logistic Regression Classifier on test data is : {:.3f}'.format(
print("----------------------------------------------------------")
```

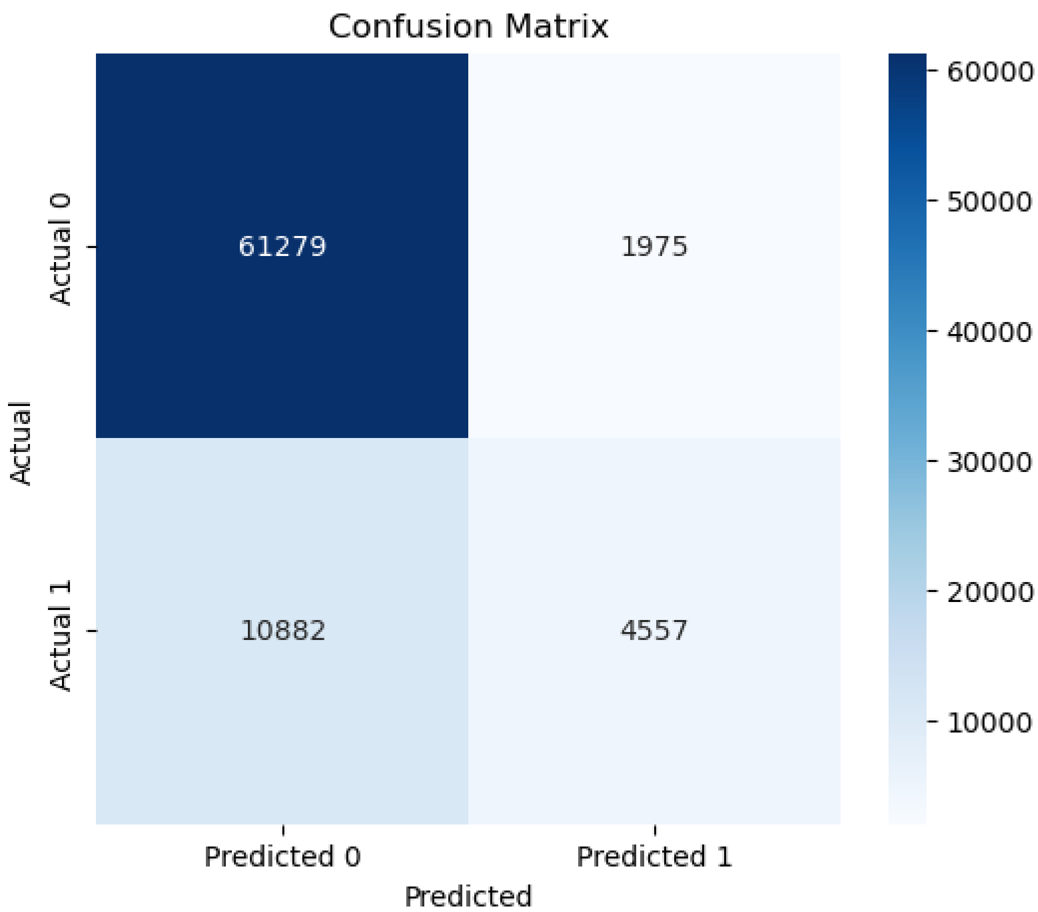
```
# Print classification metrics
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print(f'Precision: {precision_score(y_test, y_pred)}')
print(f'Recall: {recall_score(y_test, y_pred)}')
print(f'F1 Score: {f1_score(y_test, y_pred)}')
```

```
Accuracy of Logistic Regression Classifier on test data is : 0.837
---------------------------------------------------------
Accuracy: 0.8366182506703265
Precision: 0.697642375995101
Recall: 0.2951616037308116
F1 Score: 0.4148195348413818
```

1. The model is correctly classifying the target variable for approximately 83.7% of the instances in the test data.
2. The model is performing well in terms of making correct predictions across both classes.

confusion_matrix

In [75]:
```
cm = confusion_matrix(y_test,y_pred)
# Plot the confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'A
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



1. True Positives (TP): 4557

2. True Negatives (TN): 61279
3. False Positives (FP): 1975
4. False Negatives (FN): 10882

Interpretation:

1. The high number of True Negatives (61279) suggests that the model is performing well in correctly predicting instances of Class 0.
2. The True Positives (4557) indicate successful predictions of Class 1.
3. The False Positives (1975) and False Negatives (10882) represent areas where the model is making errors.

Classification Report

In [76]:
```
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.85      0.97      0.91     63254
           1       0.70      0.30      0.41     15439

    accuracy                           0.84     78693
   macro avg       0.77      0.63      0.66     78693
weighted avg       0.82      0.84      0.81     78693
```

Key Metrics:

1. *Precision*: Precision is the ratio of true positives to the sum of true positives and false positives. A higher precision value indicates fewer false positives.

2. *Recall (Sensitivity)*: Recall is the ratio of true positives to the sum of true positives and false negatives. A higher recall value indicates fewer false negatives.

3. *F1-Score*: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall.

Class 0 (Negative Class):

1. Precision: 0.85 - Out of all instances predicted as Class 0, 85% are correctly classified.
2. Recall (Sensitivity): 0.97 - Out of all actual instances of Class 0, the model correctly identifies 97%.
3. F1-Score: 0.91 - The harmonic mean of precision and recall for Class 0 is 91%.
4. Support: 63254 - The number of instances for Class 0 in the test set is 63254.

Class 1 (Positive Class):

1. Precision: 0.70 - Out of all instances predicted as Class 1, 70% are correctly classified.
2. Recall (Sensitivity): 0.30 - Out of all actual instances of Class 1, the model correctly identifies 30%.
3. F1-Score: 0.41 - The harmonic mean of precision and recall for Class 1 is 41%.
4. Support: 15439 - The number of instances for Class 1 in the test set is 15439.

Overall Metrics:

1. Accuracy: 0.84 (84%) - The overall accuracy of the model on the test set is 84%.
2. Macro Avg Precision: 0.77
3. Macro Avg Recall: 0.63
4. Macro Avg F1-Score: 0.66
5. Weighted Avg Precision: 0.82
6. Weighted Avg Recall: 0.84
7. Weighted Avg F1-Score: 0.81
8. Weighted Avg Support: 78693

Insights:

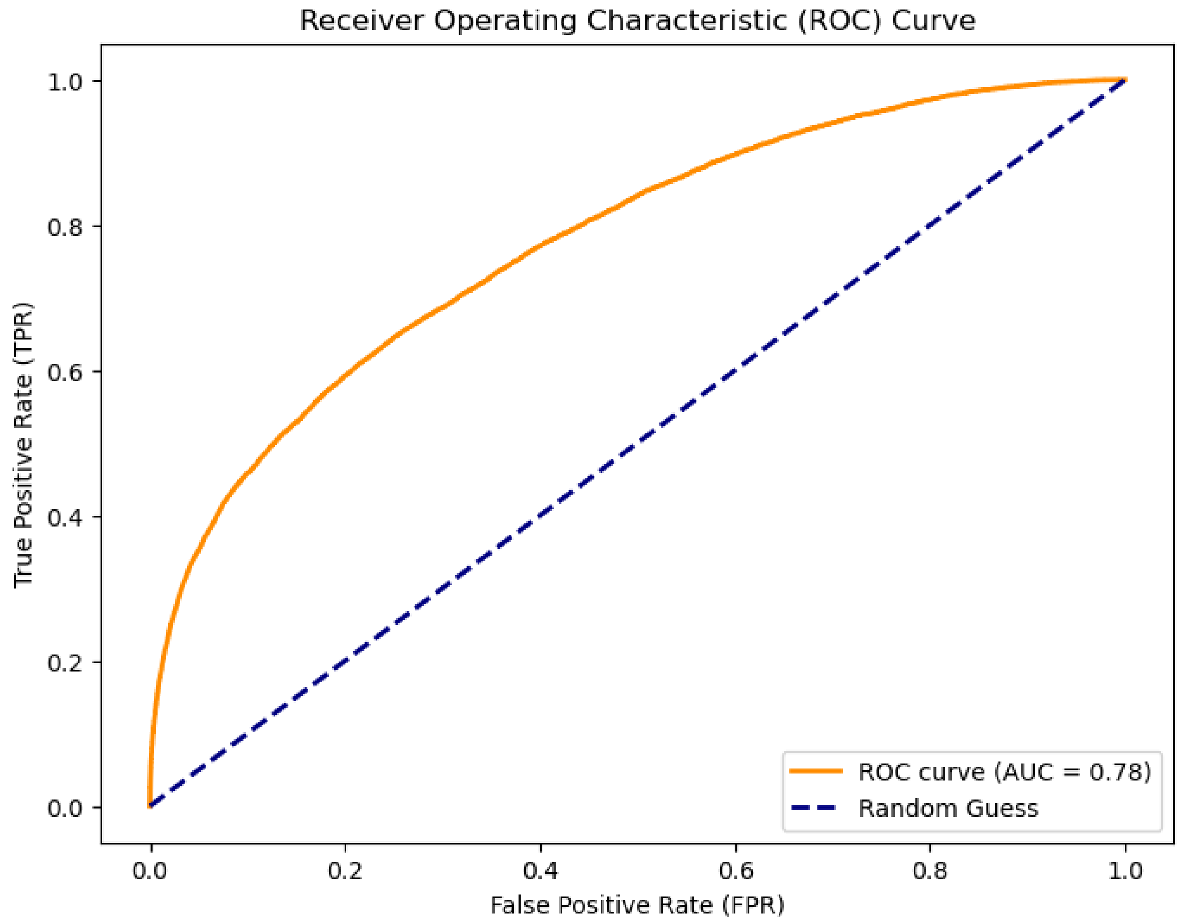
1. The model is performing well in terms of precision, recall, and F1-Score for Class 0, indicating its ability to correctly identify instances of Class 0.
2. However, the performance for Class 1 is relatively lower, as evidenced by lower precision, recall, and F1-Score. This suggests that the model struggles more with correctly classifying instances of Class 1.
3. The macro-averaged metrics consider the unweighted average of precision, recall, and F1-Score for both classes. The weighted-averaged metrics take into account the imbalance in class distribution.
4. The weighted average provides a more representative measure of overall performance, considering the number of instances in each class.

In summary, the model is good at identifying instances of Class 0 but needs improvement in correctly identifying instances of Class 1. Considerations for model improvement may include addressing class imbalance, feature engineering, or exploring different algorithms.

ROC-AUC Curve

In [77]:

```
# Make predictions on the test data
y_pred_proba = lr.predict_proba(x_test)[:, 1]
# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
# Calculate the area under the ROC curve (AUC)
roc_auc = auc(fpr, tpr)
# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.forma
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

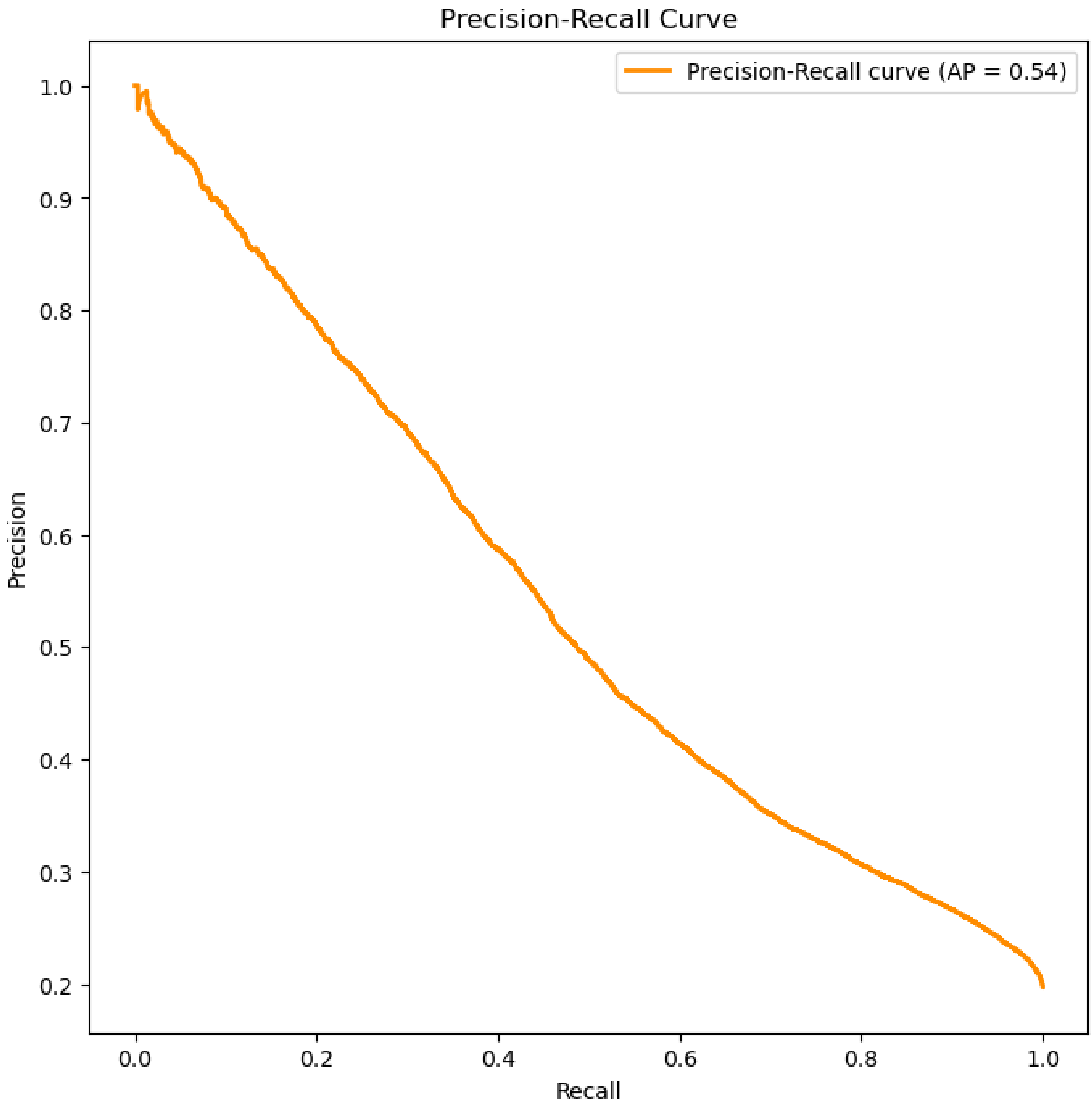


An Area Under the ROC Curve (AUC) of 0.78 is generally considered to be a moderate level of discrimination

AUC = 0.78 indicates that the model has a moderate ability to distinguish between positive and negative instances. The closer the AUC is to 1, the better the model's discrimination ability. An AUC of 0.78 suggests reasonable performance but with room for improvement.

Precision recall curve

In [78]:
```
# Calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
# Calculate the average precision
avg_precision = average_precision_score(y_test, y_pred_proba)
# Plot the Precision-Recall curve
plt.figure(figsize=(8, 8))
plt.plot(recall, precision, color='darkorange', lw=2, label='Precision-Recall curve
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right')
plt.show()
```



An Average Precision (AP) of 0.54 suggests that the Precision-Recall curve has a moderate performance.

AP is the area under the Precision-Recall curve, providing a summary measure of the model's ability to balance precision and recall across different probability thresholds. The values of AP range from 0 to 1, where 1 indicates perfect precision and recall, and 0 indicates poor performance.

Interpretation:

An AP of 0.54 suggests that the model has a moderate ability to balance precision and recall.This indicates that the model is able to identify positive instances with reasonable precision but may miss some positive instances.

6. Thinking from a bank's perspective, which metric should our primary focus be on..
1. ROC AUC
2. Precision
3. Recall
4. F1 Score

Ans: *In a banking context, where the consequences of both false positives and false negatives can be significant, F1 score or a combination of precision and recall might be a good choice. Consider the business goals, regulatory requirements, and the specific costs associated with false positives and false negatives when selecting the primary metric. It's common to evaluate models using multiple metrics and consider the overall impact on business objectives.*

Cross-validation and understanding the model's performance under different scenarios can provide a more comprehensive view of its effectiveness.

7. How does the gap in precision and recall affect the bank?

Ans: *The gap between precision and recall reflects the model's ability to balance the trade-off between false positives and false negatives. The bank should carefully consider its priorities, risk tolerance, and the specific consequences of both types of errors when making decisions about model performance and deployment.*

Tradeoff Questions and Questionnaire

Tradeoff Questions:

1. How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.

Addressing the tradeoff between minimizing false positives and ensuring the detection of real defaulters is crucial in the context of the banking industry.*

1. Avoiding Approving Risky Loans (Minimizing False Positives)
2. Avoid Potential Loan Troubles.

Since NPA (non-performing asset) is a real problem in this industry, it’s important we play safe and shouldn’t disburse loans to anyone

Insights and Recommendations

Insights

1. Logistic regression is used for binary classification, and its application is evident in this context.
2. The overall accuracy is 83.66%, suggesting that the model correctly predicts the target variable in this proportion of cases.
3. Precision is 69.76%, indicating the proportion of predicted positive instances that are truly positive. A higher precision is desirable, especially in scenarios where false positives are costly.
4. Recall is 29.52%, representing the proportion of actual positive instances that were correctly predicted by the model. A higher recall is beneficial when the cost of false negatives is high.
5. The F1 score is 41.48%, which is the harmonic mean of precision and recall. It balances the trade-off between precision and recall.

6. Confusion Matrix: TP: 4557, TN: 61279, FP: 1975, FN: 10882.
7. Macro Avg Precision, Recall, and F1-Score are provided for a broader understanding, treating each class equally. Weighted Avg metrics consider class imbalance, and they are slightly higher than macro avg due to the imbalanced dataset.
8. AUC of 0.78 indicates the model's ability to distinguish between positive and negative instances. AP of 0.54 measures the precision-recall trade-off and is useful for imbalanced datasets.

Recommendations:

1. Given the imbalance in the dataset (low recall), consider addressing class imbalance techniques such as oversampling (SMOTE) or undersampling to improve model performance on the minority class.
2. Evaluate the relevance of features and consider feature engineering techniques to enhance the model's discriminatory power.
3. Experiment with adjusting the classification threshold to balance precision and recall based on the specific goals and constraints of the application.
4. Explore more complex models or ensemble methods to capture non-linear relationships in the data.
5. Fine-tune hyperparameters of the logistic regression model to potentially improve performance.
6. If model interpretability is crucial, logistic regression is advantageous. However, if predictive performance is the primary concern, consider exploring other models.
7. Validate the model on external datasets to ensure generalizability beyond the current dataset.
8. Clearly communicate the trade-offs between precision and recall based on the specific context to stakeholders.
9. Implement continuous monitoring and updating of the model as new data becomes available.

Improvements based on recommendations:

In [79]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\gyanp\anaconda3\lib\site-packa
ges (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\gyanp\anaconda3\lib\si
te-packages (from imblearn) (0.11.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\gyanp\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\gyanp\anaconda3\li
b\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\gyanp\anaconda3\lib
\site-packages (from imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\gyanp\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: scipy>=1.5.0 in c:\users\gyanp\anaconda3\lib\site-p
ackages (from imbalanced-learn->imblearn) (1.9.1)
```

In [80]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.datasets import make_classification
```

Multicollinearity Check

In [81]:
```
def cal_vif(X):
    # Calculating the VIF
    vif=pd.DataFrame()
    vif['Feature']=X.columns
    vif['VIF']=[variance_inflation_factor(X.values,i) for i in range(X.shape[1])]
    vif['VIF']=round(vif['VIF'],2)
    vif=vif.sort_values(by='VIF',ascending=False)
    return vif
```

In [82]:
```
cal_vif(X)

# feature "revol_util" is 6.20, which is greater than 5. Hence, drop this feature.
```

Out[82]:

	Feature	VIF
4	revol_util	6.20
3	dti	4.92
1	annual_inc	4.60
0	sub_grade	4.51
5	mort_acc	2.93
6	zip_code	2.63
2	verification_status	2.04

In [83]:
```
X.drop(columns=['revol_util'],axis=1,inplace=True)
cal_vif(X)

# all features have VIF less than 5.
```

Out[83]:

	Feature	VIF
3	dti	4.34
1	annual_inc	4.25
0	sub_grade	3.94
4	mort_acc	2.92
5	zip_code	2.62
2	verification_status	2.00

Validation using KFold

In [84]:
```
# perform kfold cross validation
X=scaler.fit_transform(X)
# Create a k-fold cross-validator
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Perform k-fold cross-validation
cross_val_results = cross_val_score(lr, X, y, cv=kf, scoring='accuracy')
```

```
# Print the cross-validation results
print(f'Cross-Validation Results: {cross_val_results}')
print(f'Mean Accuracy: {cross_val_results.mean()}')
```

```
Cross-Validation Results: [0.83308553 0.83713926 0.83893104 0.83754591 0.83860064]
Mean Accuracy: 0.8370604755187883
```

```
The cross-validation results indicate a consistently high accuracy
for each fold, ranging from
approximately 83.31% to 83.89%. The mean accuracy across all folds
is approximately 83.71%, suggesting
that, on average,the model correctly predicts the target variable
with this accuracy.
```

Dealing with Imbalanced data - SMOTE (Synthetic Minority Over-sampling Technique)

In [85]:
```
# Apply SMOTE to the training data
smote = SMOTE(sampling_strategy='auto', random_state=1)
X_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)
```

In [86]:
```
# Initialize
lr_1=LogisticRegression(max_iter=1000)
# Train the model on the resampled training data
lr_1.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_1 = lr_1.predict(x_test)

# Print classification metrics
print(f'Accuracy: {accuracy_score(y_test, y_pred_1)}')
print(f'Precision: {precision_score(y_test, y_pred_1)}')
print(f'Recall: {recall_score(y_test, y_pred_1)}')
print(f'F1 Score: {f1_score(y_test, y_pred_1)}')
```

```
Accuracy: 0.7147903879633513
Precision: 0.3736243911239401
Recall: 0.6707040611438565
F1 Score: 0.4799091625341799
```

In [87]:
```
report_1 = classification_report(y_test,y_pred_1)
print(report_1)
```

```
              precision    recall  f1-score   support

           0       0.90      0.73      0.80     63254
           1       0.37      0.67      0.48     15439

    accuracy                           0.71     78693
   macro avg       0.64      0.70      0.64     78693
weighted avg       0.80      0.71      0.74     78693
```

1. The average accuracy has been decreased.
2. Recall has been increased due to balanced data
3. f1-score is also increased.

Questionnaire (The answers of all questions have also been attached with relevant line code)

1. What percentage of customers have fully paid their Loan Amount?

Ans: 80.38%

2. Comment about the correlation between Loan Amount and Installment features.

Ans: Correlation between Loan Amount and Installment features is 0.95.

1. A correlation coefficient of 0.95 indicates a very strong positive linear relationship between the two features.

2. if one feature increases, the other tends to increase almost linearly.

3. It indicate multicollinearity.

4. Both features are capturing similar information or are redundant.

3. The majority of people have home ownership as MORTGAGE.

4. People with grades 'A' are more likely to fully pay their loan. (T/F)

Ans: YES

5. Name the top 2 afforded job titles.

Ans: Teacher and Manager

6. Thinking from a bank's perspective, which metric should our primary focus be on.. 1. ROC AUC 2. Precision 3. Recall 4. F1 Score

Ans: In a banking context, where the consequences of both false positives and false negatives can be significant, F1 score or a combination of precision and recall might be a good choice. Consider the business goals, regulatory requirements, and the specific costs associated with false positives and false negatives when selecting the primary metric. It's common to evaluate models using multiple metrics and consider the overall impact on business objectives.

Cross-validation and understanding the model's performance under different scenarios can provide a more comprehensive view of its effectiveness.

7. How does the gap in precision and recall affect the bank?

Ans: The gap between precision and recall reflects the model's ability to balance the trade-off between false positives and false negatives. The bank should carefully consider its priorities, risk tolerance, and the specific consequences of both types of errors when making decisions about model performance and deployment.

8. Which were the features that heavily affected the outcome?

Ans: loan_status feature is target variable and hence conclude as outcome. This feature has maximum correlation with zip_code of 0.346973.

The correlation coefficient of 0.346973 suggests a moderate positive linear relationship between the target and the "zip_code" variable.As the feature increases, there is a tendency for the "zip_code" to increase moderately.This correlation could imply that the "zip_code" variable contains some information about the feature, or vice versa.

9. Will the results be affected by geographical location? (Yes/No)

Ans: YES, there is probability that results may affect by geographical location. As geographical location based on zip code and zip_code feature has moderate positive linear relationship with result.